- 머신러닝 : 데이터 가공 및 변환 ->> 모델 학습 및 예측 ->> 평가

1. 정확도

$$정확도 = \frac{예측결과가동일한데이터건수}{전체예측데이터건수}$$

- 단순한 알고리즘으로 예측 -> 정확도 high 일 수 도 있기 때문에 "정확도"는 조심히 사용!
- <특히> imbalanced 레이블 값 분포에서는 사용 x

In [18]:

```
#단순 알고리즘
from sklearn.base import BaseEstimator
import numpy as np

class MyDummyClassifier(BaseEstimator) :
    #fit() 매서드는 아무것도 학습x
    def fit(self, X, y=None) :
        pass

    #predict() 매서드는 Sex = 1 -> 0
    def predict(self,X) :
        pred = np.zeros((X.shape[0],1))
        for i in range(X.shape[0]) :
            if X['Sex'].iloc[i] == 1:
                pred[i] = 0
            else :
                pred[i] = 1
        return pred
```

In [19]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Null 처리 함수
def fillna(df):
    df['Age'].fillna(df['Age'].mean(),inplace=True)
    df['Cabin'].fillna('N',inplace=True)
    df['Embarked'].fillna('N',inplace=True)
    df['Fare'].fillna(0,inplace=True)
    return df

# 불필요한 속성 제거
def drop_features(df):
    df.drop(['PassengerId','Name','Ticket'],axis=1,inplace=True)
    return df

# 레이블 인코딩 수행.
def format_features(df):
    df['Cabin'] = df['Cabin'].str[:1]
    features = ['Cabin','Sex','Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

# 앞에서 설정한 Data Preprocessing 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df
```

In [20]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#타이타닉 생존자 예측
#데이터 임포트/가공/분할
titanic = pd.read_csv('titanic_train.csv')
y = titanic['Survived']
X = titanic.drop('Survived',axis =1)
X = transform_features(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state= 0)
```

In [21]:

```
#fit/predict/evaluate
myclf = MyDummyClassifier()
myclf.fit(X_train,y_train)

mypredictions=myclf.predict(X_test)
print('test accuracy : ', np.round(accuracy_score(y_test,mypredictions),5))
```

```
test accuracy :  0.78771
```

- 불균형한 데이터 셋에서 accuracy는 높게 나올 수 있기 때문에 적절한 평가 기준 x

In [22]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd

class MyFakeClassifier(BaseEstimator) :
    def fit(sel, X, y) :
        pass
    def predict(self,X) :
        return np.zeros((len(X),1),dtype = bool) # X의 크기만큼 다 False로 만듦

digits = load_digits()

#digits 번호가 7 -> True -> astype(int) -> 1 // 나머지는 False(0)
y =(digits.target ==7).astype(int)
X_train, X_test, y_train, y_test = train_test_split(digits.data , y, random_state =11)
```

In [23]:

```
#y_test 분포도 확인
print('label set size : ',y_test.shape, '₩n')
print('test label 0/1 dist ₩n' , pd.Series(y_test).value_counts())

#Dummy Classifier로 fit/predict/evaluate
fakeclf = MyFakeClassifier()
fakeclf.fit(X_train,y_train)
mypredict = fakeclf.predict(X_test)

print('test accuracy : ', accuracy_score(y_test,mypredict))
```

```
label set size :  (450,)

test label 0/1 dist
 0    405
1     45
dtype: int64
test accuracy :  0.9
```

2. 오차 행렬(Confusion Matrix)

- 이진 분류에서 많이 사용됨
- from sklearn.metrics import confusion_matrix 사용

In [24]:

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test,mypredict)
```

Out[24]:

```
array([[405,   0],
       [ 45,   0]], dtype=int64)
```

$$\text{정확도} = \frac{(TN+TP)}{(TN+TP+FN+FP)}$$

- 불균형한 데이터에서 Negative에 대한 예측 정확도로 전체 분류의 정확도가 높게 나타날 수 있다 -> 수치 판단 오류 발생 가능성 높음

> 불균형한 data set에서는 정밀도 / 재현율이 더 좋음!

3. 정밀도와 재현율

- 정밀도 & 재현율 : Positive data set의 예측 성능에 더 초점을 둠.

$$\text{정밀도} = \frac{TP}{(FP+TP)}$$
$$\text{재현율} = \frac{TP}{(FN+TP)}$$

- 정밀도 : "예측"을 Positive로 한 대상 중에 실제 = 예측이 positive로 일치하는 비율

> 스펨메일 여부 판단 모델

- 재현율 : "실제 값"이 Positive인 대상 중에 실제 = 예측이 positive로 일치하는 비율

> 재현율이 중요한 경우 : 실제 Positive인 데이터를 Negative로 잘못 판단하면 안되는 경우
> ex) 암 판단 모델, 금융 사기 적발 모델

In [25]:

```
# titanic example에 적용
# 정밀도 : precision_score()
# 재현율 : recall_score()

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

def get_clf_eval(y_test,pred) :
    confusion = confusion_matrix(y_test,pred)
    accuracy = accuracy_score(y_test,pred)
    precision = precision_score(y_test,pred)
    recall = recall_score(y_test,pred)
    print('[Confusion Matrix] ₩n₩n', confusion)
    print("₩n 정확도 : {0} , 정밀도 : {1} , 재현율 : {2}".format(accuracy,precision,recall))
```

In [26]:

```
#logistic regression -> titanic['Survived'] predict
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#data import/transform/split
titanic = pd.read_csv('titanic_train.csv')
y = titanic['Survived']
X = titanic.drop('Survived',axis = 1)
X = transform_features(X)

X_train, X_test, y_train, y_test = train_test_split(X, y , test_size =0.2, random_state = 11)

lr_clf = LogisticRegression()
lr_clf.fit(X_train,y_train)
pred = lr_clf.predict(X_test)
get_clf_eval(y_test,pred)
```

```
[Confusion Matrix]

 [[104  14]
 [ 13  48]]

 정확도 : 0.8491620111731844 , 정밀도 : 0.7741935483870968 , 재현율 : 0.7868852459
016393

C:₩Users₩1216k₩.conda₩envs₩test₩lib₩site-packages₩sklearn₩linear_model₩_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

- 재현율/정밀도를 조금 더 강화할 방법은?

> 분류의 결정 임곗값(Threshold)를 조정해서 높일 수 있다!

- Threshold 이상 : Positive
- Threshold 이하 : Negative (보통 0.5로 많이 잡음)
- predict_proba() : 개별 label의 예측 확률을 계산해줌 -> 이 값과 Threshold 비교해서 T/F 구분!

In [27]:

```
#predict_proba()를 수행한 뒤 반환 값을 확인하고 predict() 결과와 비교
pred_proba = lr_clf.predict_proba(X_test)
pred = lr_clf.predict(X_test)
print('pred_proba() 결과 shape : {0}'.format(pred_proba.shape))
print('₩n pred_proba 3 samples : ₩n {0}'.format(pred_proba[:3]))

# pred_proba array & pred array -> concatenate
pred_proba_result = np.concatenate([pred_proba,pred.reshape(-1,1)], axis = 1)
print('₩n',pred_proba_result[:3])
#pred_proba_result[0][0][0] + pred_proba_result[0][0][1] = pred_proba_result[0][0][2]
```

```
pred_proba() 결과 shape : (179, 2)

 pred_proba 3 samples :
 [[0.4616653  0.5383347 ]
 [0.87862763 0.12137237]
 [0.87727002 0.12272998]]

 [[0.4616653  0.5383347  1.        ]
 [0.87862763 0.12137237 0.        ]
 [0.87727002 0.12272998 0.        ]]
```

In [28]:

```
#Binarizer의 fit_transform() : ndarray value < threshold 이면 return 0, else return 1.
from sklearn.preprocessing import Binarizer

X = [[1,-1,2],
    [2,0,0],
    [0,1.1,1.2]]

#X의 value를 threshold = 1.1을 기준으로 0/1 나누기
binarizer = Binarizer(threshold=1.1)
print(binarizer.fit_transform(X))
```

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

In [29]:

```
#logistic regression -> predict_proba를 threshold = 0.5로 Binarizer 적용하여 최종 예측값 구하기
from sklearn.preprocessing import Binarizer

#threshold
custom_threshold = 0.5

#predict_proba() 반환값의 두 번째 칼럼(= Positive class)만 추출해 Binarizer 적용
pred_proba_1= pred_proba[:,1].reshape(-1,1)

binarizer = Binarizer(threshold = custom_threshold).fit(pred_proba_1)
custom_predict = binarizer.transform(pred_proba_1)

get_clf_eval(y_test,custom_predict)
```

```
[Confusion Matrix]

 [[104  14]
 [ 13  48]]

 정확도 : 0.8491620111731844 , 정밀도 : 0.7741935483870968 , 재현율 : 0.7868852459
016393
```

- Threshold를 낮추면?

> 재현율↑ 정밀도↓. 분류 결정 임계값이 Positive predict value를 결정하는 확률의 기준이 되기 때문!

In [30]:

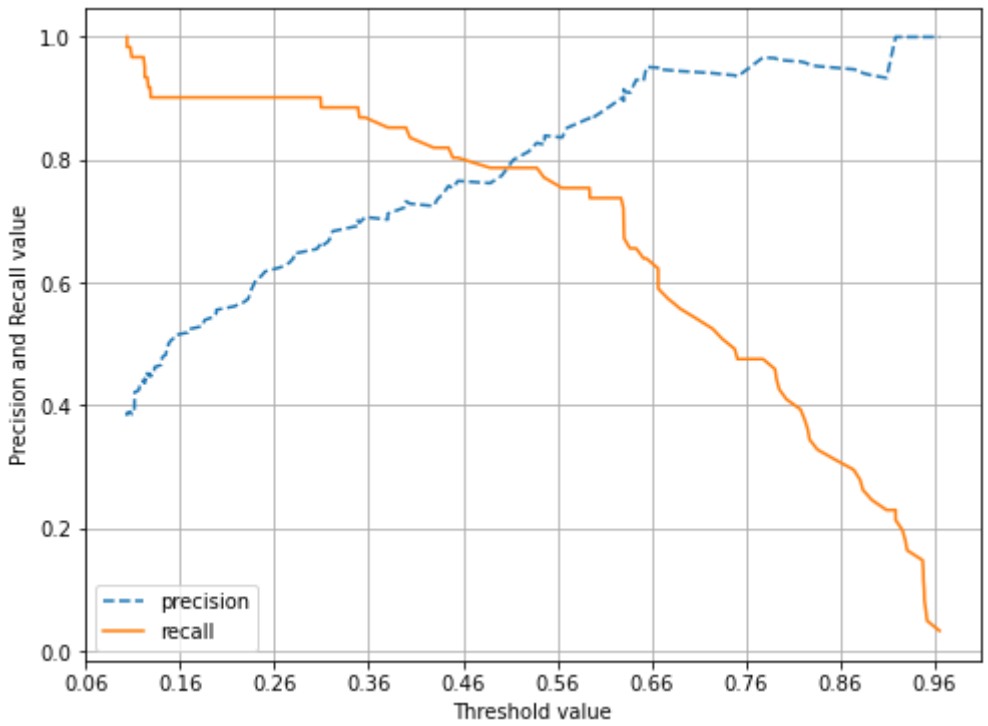
```
from sklearn.metrics import precision_recall_curve

# 레이블 값이 1일때의 예측 확률을 추출
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

# 실제값 데이터 셋과 레이블 값이 1일 때의 예측 확률을 precision_recall_curve 인자로 입력
precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_class1 )
print('반환된 분류 결정 임곗값 배열의 Shape:', thresholds.shape)
print('반환된 precisions 배열의 Shape:', precisions.shape)
print('반환된 recalls 배열의 Shape:', recalls.shape)

print("thresholds 5 sample:", thresholds[:5])
print("precisions 5 sample:", precisions[:5])
print("recalls 5 sample:", recalls[:5])

#반환된 임계값 배열 로우가 147건이므로 샘플로 10건만 추출하되, 임곗값을 15 Step으로 추출.
thr_index = np.arange(0, thresholds.shape[0], 15)
print('샘플 추출을 위한 임계값 배열의 index 10개:', thr_index)
print('샘플용 10개의 임곗값: ', np.round(thresholds[thr_index], 2))

# 15 step 단위로 추출된 임계값에 따른 정밀도와 재현율 값
print('샘플 임계값별 정밀도: ', np.round(precisions[thr_index], 3))
print('샘플 임계값별 재현율: ', np.round(recalls[thr_index], 3))
```

```
반환된 분류 결정 임곗값 배열의 Shape: (143,)
반환된 precisions 배열의 Shape: (144,)
반환된 recalls 배열의 Shape: (144,)
thresholds 5 sample: [0.10393302 0.10393523 0.10395998 0.10735757 0.10891579]
precisions 5 sample: [0.38853503 0.38461538 0.38709677 0.38961039 0.38562092]
recalls 5 sample: [1.         0.98360656 0.98360656 0.98360656 0.96721311]
샘플 추출을 위한 임계값 배열의 index 10개: [  0  15  30  45  60  75  90 105 120 13
5]
샘플용 10개의 임곗값:  [0.1  0.12 0.14 0.19 0.28 0.4  0.57 0.67 0.82 0.95]
샘플 임계값별 정밀도:  [0.389 0.44  0.466 0.539 0.647 0.729 0.836 0.949 0.958 1.
]
샘플 임계값별 재현율:  [1.    0.967 0.902 0.902 0.902 0.836 0.754 0.607 0.377 0.14
8]
```

- 임계값, 정밀도는 동시에 ↑ but 정밀도↓

In [31]:

```
#임계값에 따른 정밀도 & 재현율 curve
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline

def precision_recall_curve_plot(y_test , pred_proba_c1):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarray 추출.
    precisions, recalls, thresholds = precision_recall_curve( y_test, pred_proba_c1)

    # X축을 threshold값으로, Y축은 정밀도, 재현율 값으로 각각 Plot 수행. 정밀도는 점선으로 표시
    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary],label='recall')

    # threshold 값 X 축의 Scale을 0.1 단위로 변경
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    # x축, y축 label과 legend, 그리고 grid 설정
    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recall value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot( y_test, lr_clf.predict_proba(X_test)[:, 1] )
```



4. F1 스코어

1. F1 스코어 : 정밀도와 재현율을 결합한 지표(두 지표 값이 어느 한쪽으로 치우치지 않았을 때 높은 값을 가짐)

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precisio}} = 2 * \frac{precision * recall}{precision + recall}$$

In [32]:

```
from sklearn.metrics import f1_score

f1 = f1_score(y_test , pred)
print('F1 스코어: {0:.4f}'.format(f1))
```

F1 스코어: 0.7805

5. ROC curve & AUC

1. ROC 곡선과 AUC : 이진 분류의 예측 성능 측정에서 중요하게 사용됨

- ROC 곡선 : FPR(X룩)이 변할 때 TPR(Y축)이 어떻게 변하는지를 나타내는 곡선

> TPR = True Positive Rate =TP/(FN+TP) =재현율 = Sensitivity(민감도) , TNR = True Negative Rate = TN/(FP+TN) = Specificity(특이성)

- FPR = FP/(FP+TN) = 1 - TNR = 1- Specificity
- AUC : ROC 곡선 밑의 면적, 1에 가까울수록 좋은 수치.

> AUC를 키우려면? FPR이 작은 상태에서 얼마나 큰 TPR을 갖는 지가 관건.

In [34]:

```
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},W
          F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```